

EV368629852

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

APPLICATION FOR LETTERS PATENT

PROVIDING AVAILABILITY INFORMATION USING A
DISTRIBUTED CACHE ARRANGEMENT AND UPDATING THE CACHES USING
PEER-TO-PEER SYNCHRONIZATION STRATEGIES

Inventor:

Richard M. Hawes

ATTORNEY'S DOCKET NO. MS1-1840US

TECHNICAL FIELD

This invention relates to notifying a user of the unavailability of an application provided by a data processing system. This invention also relates to a strategy for updating information in a data processing system, and, in a more particular implementation, to a strategy for updating availability information in a data processing system that includes multiple server modules.

BACKGROUND

Companies and other organizations commonly provide resources over the Internet using a cluster of servers, referred to as a server farm. A server farm may allocate different servers to handle respective processing tasks or services. In this case, the server farm can route a user's request to an appropriate server that performs the service referenced by the user's request. Alternatively, or in addition, the server farm may allocate a group of redundant servers for handling the same task. In this case, load balancing functionality provided by the server farm can dynamically determine which server in the group of redundant servers is best able to handle the user's request at a particular point in time. In general, different processing environments can rely on different resource and processing allocation schemes to best service the requests of its users, depending on the unique characteristics of these different processing environments.

Fig. 1 shows a high-level architecture associated with an exemplary server farm 102. The server farm 102 includes an exemplary collection of servers (104, 106, 108, ... 110). Each of these servers (104, 106, 108, ... 110) has access to a collection of resources (112, 114, 116, ... 118). The resources (112, 114, 116, ... 118) loosely represent any processing functionality and/or information used to respond to users' requests. To facilitate illustration, the resources (112, 114, 116, ... 118) are shown as distinct entities

1 that are coupled to respective servers (104, 106, 108, ... 110). However, more broadly,
2 the resources (112, 114, 116, ... 118) can represent any functionality and/or information
3 provided at any location. For instance, the resources A 112 accessible to server A can
4 refer to processing functionality and/or information actually implemented on server A
5 104 itself. In one implementation, the functionality and/or information in resources A
6 112 can be entirely unique and non-overlapping with respect to the functionality and/or
7 information in resources B 114. In another implementation, certain functionality and/or
8 information in resources A 112 can be common to that of resources B 114; this
9 commonality can be achieved by either the storage of redundant copies of the same
10 resources, or by access to a single copy of shared common resources.

11 In any case, users (120, 122, ... 124) can access the resources (112, 114, 116, ...
12 118) via a wide area network (WAN) packet network (126) (such as the Internet) via
13 respective client modules (128, 130, ... 132) (such as respective personal computers or
14 application-specific processing devices). For instance, Fig. 1 shows an exemplary user
15 122 accessing the resources (112, 114, 116, ... 118) of the server farm 102 using client
16 module B 130. This request is denoted by arrow 134. The server farm 102 can route the
17 request 134 to an appropriate server (or servers) and associated resources depending on
18 the nature of the request 134 and/or based on load-balancing considerations. In the
19 illustrative and exemplary case of Fig. 1, the server farm 102 determines that server C
20 108 and associated resources C 116 should handle the request 134. A response will be
21 formulated using resources C 116 and sent back to the user 122 over the WAN 126.

22 However, server farms sometimes fail to provide the services requested by their
23 users. There are many causes of this failure. In one case, there may be an equipment
24 malfunction associated with the servers (104, 106, 108, ... 110) and/or associated
25 resources (112, 114, 116, ... 118). In another case, parts of (or the entirety of) the server

1 farm 102 can be brought offline to perform maintenance or service upgrades, etc.
2 Whatever the case, the users (120, 122, ... 124) cannot interact with the server farm 102
3 in the normal manner during these down times. In a common scenario, the users (120,
4 122, ... 124) will receive various errors messages (such as error message 136) when they
5 submit requests to the server farm 102. The server farm 102 itself can generate these
6 error messages by detecting the inability to access the needed resources. Alternatively,
7 some other entity (such as an Internet Service Provider) can provide these messages on
8 behalf of the server farm 102. For instance, this other entity can generate a failure
9 message if the server farm 102 fails to respond within a predetermined period of time
10 (such as 30 seconds).

11 The above solutions to error conditions are not fully satisfactory. For instance,
12 the user 122's request may have required the user 122 to perform various time-consuming
13 actions, such as typing in various data. If the request cannot be processed by the server
14 farm 102 then this effort is wasted and must be repeated when the server farm 102
15 becomes operational. For instance, consider the exemplary case in which the user 122 is
16 relying on the server farm 102 to transmit the user 122's email message. This message
17 may take an appreciable amount of time to type in. If the server farm 102 cannot process
18 this email because of equipment failure (or because of any other reason), the email may
19 be "lost," thus requiring the user 122 to retype this message at a later time when the
20 server farm 102 is operational again. Needless to say, such a failure in the server farm
21 102 can frustrate the user 122.

22 Accordingly, there is an exemplary need for a more effective strategy for alerting
23 users to the unavailability of system resources in a server farm including multiple servers.
24 There is another need for maintaining the integrity (e.g., currency) of information in any
25 system which implements such a strategy. However, the above-described needs are

1 merely representative of many kinds of deficiencies in existing systems that rely on
2 multiple processing modules to provide services; accordingly, more broadly stated, there
3 is an exemplary need for more effective strategies for providing services using a system
4 including multiple processing modules.

6 **SUMMARY**

7 According to one exemplary implementation, a method is described for advising a
8 user of the availability of an application in a system including plural server modules. The
9 method includes: (a) receiving, at a server module in the system, a user's request for an
10 application; (b) consulting an application store associated with the application to
11 determine whether the application is unavailable, and, if so generating a response; and (c)
12 forwarding the response to the user, wherein each of the plural server modules in the
13 system maintains its own respective application store.

14 According to another exemplary implementation, a method is described for
15 synchronizing a system including plural server modules. The method includes: (a)
16 receiving notification information at a first server module regarding a change in the
17 system; (b) acting on the notification information in the first server module; and (c)
18 propagating the notification information from the first server module to at least a second
19 server module. The notification information can pertain to an indication of whether or
20 not at least one application used by the system is available to service user requests.

21 According to another exemplary implementation, another method is described for
22 synchronizing a system including plural server modules. The method includes: (a)
23 sending first status information reflecting a state in a first server module to a second
24 server module; (b) merging the first status information with second status information,
25 the second status information reflecting a state of the second server module, to produce

merged information; (c) sending the merged information from the second server module to the first server module; and (d) acting on the merged information at the first server module. The status information can include notification information regarding a change in the system. More specifically, the notification information can comprise an indication of whether or not at least one application used by the system is available to service user requests.

Additional implementations and features will be described in the following.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 shows a conventional system for providing services to users using a server farm.

Fig. 2 shows a system for providing services to users using a server farm including multiple application caches and synchronization functionality.

Fig. 3 shows the use of the synchronization functionality illustrated in Fig. 2 to synchronize the system when a server module returns to service after having been inactive for any reason.

Fig. 4 shows an exemplary procedure for notifying users of the unavailability of resources using the multiple application caches shown in Fig. 2.

Fig. 5 shows an exemplary procedure for disseminating availability information throughout the system shown in Fig. 2.

Fig. 6 shows an exemplary procedure for synchronizing a starting server module with the rest of the system of Fig. 2 when the starting server module returns to service after having been inactive for any reason.

1 Fig. 7 complements Fig. 6 by showing how a reference server module interacts
2 with the starting server module to assist the starting server module to synchronize itself
3 with respect to the rest of the system.
4

5 The same numbers are used throughout the disclosure and figures to reference like
6 components and features. Series 100 numbers refer to features originally found in Fig. 1,
7 series 200 numbers refer to features originally found in Fig. 2, series 300 numbers refer
8 to features originally found in Fig. 3, and so on.
9

10 **DETAILED DESCRIPTION**

11 Any of the functions described herein can be implemented using software,
12 firmware (e.g., fixed logic circuitry), manual processing, or a combination of these
13 implementations. The term “logic” or “module” as used herein generally represents
14 software, firmware, or a combination of software and firmware. For instance, in the case
15 of a software implementation, the term “logic” or “module” represents program code that
16 performs specified tasks when executed on a processing device or devices (e.g., CPU or
17 CPUs). The program code can be stored in one or more computer readable memory
18 devices. The illustrated separation of logic and modules into distinct units may reflect an
19 actual physical grouping and allocation of such software and/or hardware, or can
20 correspond to a conceptual allocation of different tasks performed by a single software
21 program and/or hardware unit. The illustrated logic and modules can be located at a
22 single site (e.g., as implemented by a single processing device), or can be distributed over
23 plural locations.

24 To provide concrete examples, the synchronization strategies are described in the
25 specific context of a system which updates availability information among multiple

1 server modules. The availability information indicates whether applications that can be
2 accessed by the server modules are available or unavailable. However, the
3 synchronization strategies can be employed in other kinds of systems to update other
4 kinds of information.

6 **A. Exemplary Architecture**

7 *Overview of Architecture*

8 Fig. 2 shows a system 200 including a server farm including plural server
9 modules (202, 204, 206, ... 208). As described above, a server farm refers to a collection
10 or a cluster of server modules (and associated equipment). The server modules (202,
11 204, 206, ... 208) can be located at a single site or distributed over plural sites.

12 The term "server module" itself can refer to any functionality for providing
13 services to a user. In one exemplary case, a "server module" corresponds to a server-type
14 programmable computer. The server-type programmable computer can include one or
15 more processors, volatile and non-volatile memory for storing machine readable code and
16 other information, media access functionality, interface functionality, etc. In another
17 case, the term "server module" can refer to separate code modules implemented by a
18 single server-type programmable computer, each for implementing server functionality.
19 In another case, the term "server module" can pertain to, in whole or in part, fixed (e.g.,
20 non-programmable) logic circuitry for implementing server functionality. Still other
21 implementations can be provided.

22 Each server module (202, 204, 206, ... 208) has access to various resources (210,
23 212, 214, ... 216) used to perform some service. Resources (210, 212, 214, ... 216) can
24 include processing functionality (e.g., machine readable code, processing equipment,
25 etc.), information (e.g., database records), or other machine readable "objects." As in the

1 case described for the system 100 of Fig. 1, the coupling between server modules (202,
2 204, 206, ... 208) and resources (210, 212, 214, ... 216) should be construed generally to
3 encompass multiple different interpretations and corresponding implementations.
4 Generally, Fig. 2's labeling of separate resource blocks with separate letters (i.e., A, B, C,
5 etc.) can denote separate and non-overlapping groups of resource items, but need not.
6 Consider, for example, the representative case in which server module A 202 has access
7 to resources A 210, and server module B 204 has access to resources B 212. In this case,
8 resources A 210 can refer to a set of resource items that is unique and non-overlapping
9 with respect to the set of resource items associated with resources B 212. Alternatively, a
10 subset of resource items in resources A 210 can also be present in resources B 212, or
11 indeed, all of the resource items in resources A 210 can be present in resources B 212
12 (that is, in this example, resources A 210 and resources B 212 describe the same set of
13 resource items). This commonality can be achieved by providing two separate copies of
14 the common resource items corresponding to resources A 210 and resources B 212,
15 respectively. Alternatively, this commonality can be implemented by providing a single
16 copy of the shared resource items that is accessible to server module A 202 and server
17 module B 204, respectively. Finally, the resources (210, 212, 214, ... 216) are shown as
18 coupled to and separate from respective server modules (202, 204, 206, ... 208);
19 however, the resources (210, 212, 214, ... 216) can also refer, in whole or in part, to
20 resource items that are implemented by the server modules (202, 204, 206, ... 208)
21 themselves.

22 Further, although only one tier (level) of server modules (202, 204, 206, ... 208)
23 is shown in Fig. 2, the system 200 can include multiple tiers (levels) of server modules.
24 For instance, a first company can maintain a first tier of server modules which use a
25 second tier of server modules maintained by another company or maintained by a whole

1 series of partner companies. This second tier of server modules can be viewed as
2 resources of the first tier of server modules.

3 The types of services (and associated resources) provided by the system 200 can
4 vary depending on the specific processing environment in which the system 200 is
5 employed. A representative and non-limiting list of possible services include: email
6 handling services; chat room services; information searching services; electronic
7 commerce (e.g., online shopping) services; parental control services; web page creation
8 services; help information services; online television user interface services; messenger-
9 type services; information center services pertaining to various topics, etc. For example,
10 the system 200 can implement, in whole or in part, the services provided by Microsoft
11 Corporation's MSN network or MSN TV network (provided by Microsoft Corporation of
12 Redmond, Washington). Again, these implementations pertain to only a representative
13 sampling of possible services that the system 200 can provide.

14 In any case, a user 218 can access the services provided by system 200 using
15 exemplary requesting module 220 via a wide area network (WAN) 222, such as the
16 Internet. The Transmission Control Protocol (TCP) and the Internet Protocol (IP) can be
17 used to couple the requesting module 220 to the server modules (202, 204, 206, ... 208)
18 in the system 200. However, other networks can be used, such as an intranet, a local area
19 network (LAN), and so forth. A WAN (e.g., the Internet, etc.) or some other local
20 network (e.g., a LAN, point-to-point connectivity, etc.) can be used, in whole or in part,
21 to implement communication between server modules (202, 204, 206, ... 208) in the
22 system 200. The requesting module 220 itself can be implemented by any kind of
23 computer (e.g., a general purpose computer), any kind of application specific device (e.g.,
24 a specific kind of set-top box that couples to a user's television, etc.), and so on. The
25 illustrated single user 218 and associated requesting module 220 are merely

1 representative of many such users and associated requesting modules (not shown) that
2 can interact with the system 200 via the WAN 222.

3 *Functionality for Determining Application Availability*

4 Each of the server modules (202, 204, 206, . . . 208) can implement one or more
5 “applications.” Applications may pertain to different functionality that, taken together,
6 perform a service provided by the server modules (202, 204, 206, ... 208). For instance,
7 these different applications may represent different code modules performing different
8 respective collections of tasks that, taken together, perform a service. Alternatively, the
9 applications can be selectively invoked by the user 218 without necessarily performing
10 other applications. Each of the applications may draw from all or some of the resources
11 (210, 212, 214, 216 ...). For instance, an application that provides a telephone directory
12 lookup may draw from a resource that constitutes a list of names and telephone numbers.

13 In any case, at any time, an application may have an available status or an
14 unavailable status. An application is available if it can be used to provide services to the
15 user 218 upon request. An application is unavailable if it cannot provide these services
16 for any number of reasons. One reason that an application can be unavailable is because
17 the equipment that implements the application and/or its associated resources are not
18 working properly. Another reason that an application can be unavailable is that the
19 machine readable code that implements the application is not operating properly.
20 Another reason that an application can be unavailable is that the resources (e.g., database
21 records or other “objects”) that are used by the application are not working properly.
22 Another reason that an application can be unavailable is that the application and/or its
23 associated resources have been brought “off line” for maintenance, testing or upgrading.
24 These reasons are merely representative and exemplary; the unavailability of the
25 applications may be attributed to other reasons.

Each server module (202, 204, 206, ... 208) can include one or more application stores. In one implementation, these application stores can be implemented as caches, and will be referred to from this point on as caches. (A cache refers to an in-memory store that can be quickly accessed by a referencing source.) For instance, server module A 202 includes application caches 224, 226, ... 228, while server module B 204 includes application caches 230, 232, ... 234. Each of the application caches (224, ... 234) is associated with a particular application (not shown) used by the server modules (202, 204, 206, ... 208). More specifically, the application caches (224, ... 234) can be implemented "in process" with respect to their associated applications to facilitate interaction between the applications and their respective application caches (224, ... 234).

In one implementation, the application caches (224, ...234) store availability information. The availability information indicates, for each application, whether the application is available to user requests or is not available to user requests. Accordingly, in one implementation, this availability information can be conveyed, for each application cache (224, ... 234) by a binary value, e.g., having a value of 1 (for available) and a value of 0 (for not available). In another implementation, the availability information can include additional data. For instance, if the application is not available, the availability information can convey information that specifies the reasons for this unavailability. For instance, different codes can be assigned to different reasons for unavailability, such as: "0" signifying that the cause of the unavailability is unknown; "1," indicating that the application is unavailable because of planned maintenance (where the availability information can also convey the expected downtime duration); "2," indicating that the application is unavailable because of an unplanned outage; "3," indicating that the application is unavailable because testing is being performed on the

1 application, and so on. Also, in the case of unavailability due to application failure, the
2 availability information can provide a code that identifies the component of the server
3 farm that is attributed to the failure (that is, assuming that the source of the failure can be
4 determined). The above-specified categories are merely exemplary reasons that may
5 explain the cause of a failure. The availability information can be tailored to suit the
6 characteristics of different processing environments.

7 Each server module (202, 204, 206, ... 208) includes application availability
8 determination logic (236, 238, ...), referred to for brevity as AAD logic (236, 238, ...).
9 The role of the AAD logic (236, 238, ...) is to receive the user 218's request and to
10 determine whether that request can be satisfied by the system 200. The AAD logic (236,
11 238, ...) performs this role by accessing the appropriate application cache(s) (224, ...
12 234) to read the availability information contained therein. More specifically, when the
13 user 218 makes a request, front end processing functionality (not shown) provided by the
14 system 200 can determine what server module (202, 204, 206, ... 208) is allocated to
15 servicing this request. This can be determined on the basis of addressing information in
16 the request itself, and/or based on load-balancing considerations. When an appropriate
17 server module (202, 204, 206, ... 208) receives the request, its associated AAD logic
18 (236, 238, ...) then determines what application (or applications) it invokes. (This can be
19 based on an analysis of information in the request itself, and/or based on other
20 considerations). It then accesses the appropriate application cache (224, ... 234) to read
21 the availability information contained therein.

22 Providing that the appropriate application cache (224, ... 234) indicates that the
23 application is available, then the AAD logic (236, ... 238) can instruct the appropriate
24 server module functionality and associated resources to perform that application. But in
25 the case that the application cache (224, ... 234) indicates that the application is not

1 available, then the AAD logic (236, 238, ...) can generate an error message for
2 transmission to the user 218. The error message can alert the user 218 to the fact that the
3 application is unavailable. It can further convey any of the supplemental information
4 contained in the appropriate application cache (224, ... 234), such as an indication of why
5 the application is unavailable, and an indication of when the application may become
6 available.

7 The server modules (202, 204, 206, ... 208) preferably invoke their respective
8 AAD logic (236, 238, ...) immediately after a request from the user 218 is received. This
9 provides the user 218 with an error message (in the event that an application is not
10 available) soon after the user 218 makes initial contact with the system 200. This also
11 alleviates some of the problems described in the Background section. For instance, if the
12 user 218 is using the system 200 to send an email, then the relevant server module (202,
13 204, 206, ... 208) will inform the user 218 immediately of the unavailability of the email
14 service. This will prevent the user 218 from spending time composing the email to only
15 later find out that the email service is unavailable, e.g., when he or she attempts to send it.

16 Also, the allocation of separate application caches (224, ... 234) to separate
17 applications yields other benefits. Namely, because the server modules (202, 204, 206,
18 ... 208) store the availability information locally in caches, this information can be
19 retrieved very quickly (as opposed to storing this information at a central location in a
20 non-volatile storage media, such as a disk storage). Further, allocating separate
21 availability information on an application-by-application basis allows the system 200 to
22 provide fine-grained availability information to the user 218, and also potentially allows
23 the system 200 to continue to provide services even though some of its applications and
24 associated resources may be unavailable; in marked contrast; some systems grant access
25 to their services on an all-or-nothing system-wide basis.

1 The above-identified advantages are merely exemplary and do not represent all of
2 the potential benefits that the distributed cache scheme may yield when applied to
3 different technical environments.

4 *Functionality for Disseminating Availability Information to the Distributed*
5 *Application Caches*

6 The availability of applications can change over time. Accordingly, the system
7 200 provides a mechanism for disseminating changes in availability information to the
8 various application caches (224, ... 234) used in the system 200. This functionality is
9 described below.

10 To begin with, the system 200 can learn of changes in the availability of its
11 applications (224, ... 234) (and associated resources) through different means. In one
12 technique a human operator 240 can manually input information regarding changes in the
13 availability of the applications and associated resources. This operator 240 may represent
14 an administrative person assigned the role of monitoring the system 200, or may be a
15 programmer who is performing maintenance, upgrading or testing of equipment or
16 machine readable code used by the system 200. The operator 240 can use a change-in-
17 availability notification module 242 (referred to below as a “notification module” for
18 brevity) to forward the availability information to the system 200. In one
19 implementation, the notification module 242 can be implemented as any kind of
20 computing device (such as a general purpose computer) having a suitably configured user
21 interface (not shown) for soliciting the availability information from the operator 240.

22 In another case, the system 200 may employ various kinds of monitoring
23 equipment and infrastructure for automatically monitoring the availability of different
24 applications and resources provided by the system. This equipment and infrastructure is
25 generally denoted in Fig. 2 as system monitoring functionality 244. The system

1 monitoring functionality 244 can monitor the availability of hardware used to implement
2 the applications and resources (e.g., to determine whether this hardware is powered on
3 and functioning properly), and/or can provide various software monitoring mechanisms
4 to determine whether the software used by the applications and resources is running
5 properly. In the latter case, these software monitoring mechanisms can operate using
6 various scripts, rules, counters, testing patterns, etc. The system monitoring functionality
7 244 can also interface with the system 200 via the notification module 242, or via some
8 other module.

9 The notification module 242 can transmit the notification information to one of
10 the server modules (202, 204, 206, ... 208) of the system 200 using any protocol, such as
11 a message queuing protocol. Message queuing protocols (such as MSMQ provided by
12 Microsoft Corporation) enable communication across potentially heterogeneous systems
13 and networks, parts of which may, at any given time, be temporarily offline. In this
14 protocol, applications transmit messages to respective queues and retrieve message from
15 respective queues. In the present case, messages sent using the message queuing protocol
16 can include information expressed in the Extensible Markup Language (XML), or some
17 other format. However, other protocols and mechanisms can be used to transmit
18 notification information, such as the Simple Object Access Protocol (SOAP). SOAP
19 provides a lightweight protocol to transfer information over networks or other kinds of
20 distributed environments. This protocol provides an extensible messaging framework
21 using XML to provide messages that can be sent on different kinds of underlying
22 protocols. Each SOAP message includes a header block and a body element.

23 In general, the message transmitted to the system 200 can identify the application
24 (or applications) that it pertains to. For instance, in one case, the message may convey
25 that the unavailability status affects the entire system 200 and all of its applications; in

1 another case, the message may indicate that the unavailability status affects only one
2 application, or a small number of applications. Alternatively, the message can identify
3 resources that are used by the applications, the availability of which affects the
4 availability of the applications that use these resources. The message can also contain a
5 timestamp which reflects when it was created and/or transmitted. This timestamp
6 determines the message's priority; that is, later submissions regarding the same
7 application override earlier submissions. Further, each message can contain additional
8 information, such as a code which reflects a reason that the application is unavailable (if
9 the application is assessed to be unavailable), an ID assigned to an operator who created
10 the entry (if an operator 240 created it, as opposed to the system monitoring functionality
11 244), a time when the application is expected to go down, a time when the application is
12 expected to become available again, and so on. Further still, the message can contain
13 criteria which limit the applicability of the availability information to only certain kinds
14 of requests made by the user 218, or other qualifications which make the applicability of
15 the availability information conditional on other factors. Thus, a targeted application can
16 be available for one kind of request, but unavailable for another kind of request. Still
17 further fields of information can be conveyed by the message.

18 By way of overview, the system 200 disseminates the availability information in
19 the request by sending it first to an initial server module. Assume, for the sake of
20 discussion, that this initial server module corresponds to server module A 202. This
21 initial distribution is illustrated in Fig. 2 as path 246. The initial server module A 202
22 then propagates the availability information to all of the other server modules (204, 206,
23 ... 208) in the system 200. Fig. 2 illustrates this propagation by the plural paths 248
24 linking the server module A 202 to the other server modules (204, 206, ... 208).
25

1 The system 200 can use a variety of different techniques to determine which
2 server module should act as the initial server module (which in this case is server module
3 A 202). In one case, the system 200 can rely on load-balancing functionality (not shown)
4 to determine which server module is best able to perform the role of the initial server
5 module. For instance, the load-balancing functionality may assign this task to the server
6 module currently handling the least amount of processing load. In another case, the
7 system 200 may assign the role of initial server module to a predetermined server
8 module, or a predetermined list of server modules; in the latter case, the system 200 can
9 be configured to attempt to assign the role to the first server module in the list, and if this
10 server module is unavailable, then the system 200 will attempt to assign the role to the
11 second server module in the list, and so on). Still other techniques can be used to assign
12 the role of the initial server module to one of the server modules in system 200.

13 Each server module includes functionality for receiving availability information
14 and for acting on that availability information in the manner to be described below. This
15 functionality is implemented by synchronization modules (250, 252, ...). That is server
16 module A 202 provides synchronization module 250, while server module B 204 provides
17 server module 252, etc. (The internal functionality of other server modules is not shown
18 in Fig. 2 to simplify this drawing). Server module 250 includes a message queue 254, a
19 repeater module 256, and merge logic 258. Server module 252 includes the same
20 features, namely, a message queue 260, a repeater module 262, and merge logic 264.
21 Other server modules (e.g., server module 206, ... 208) include similar features, although
22 not shown.

23 The function of the message queues (254, ... 260) is to transport messages
24 containing availability information according to a message queuing protocol. Generally,
25 the receiving server module can order these messages according to their respective

1 timestamps (which is related to the time that they were created and/or transmitted to the
2 system 200). Also, the receiving server module can be configured so as to grant priority
3 to certain messages based on various factors. For instance, certain messages can pertain
4 to critical applications; accordingly the position of these messages in the queue can be
5 taken "out of order" (that is, out of order with respect to the messages' timestamps). Still
6 other types of priority schemes can be implemented depending on the processing
7 environment in which the system 200 is employed.

8 The repeater modules (256, ... 262) serve the function of storing the received
9 availability information in files and propagating the availability information to various
10 destinations via the messaging queues (254, ... 260). For instance, in the case of the
11 initial server module A 202, the repeater module 256 can transfer the availability
12 information received via the message queue 254 into the appropriate application caches
13 (224, 226, ... 228) depending on the respective applications that the availability
14 information pertains to. This is illustrated by the arrow that points from the message
15 queue 254 to the repeater module 256, and the arrows that then point from the repeater
16 module 256 to the respective application caches (224, ... 228). The initial server module
17 A 202 also uses the repeater module 256 to transfer the availability information to the
18 respective message queues (260, ...) of the other server modules (204, 206, ... 208) via
19 the message queue 254. Fig. 2 represents this information transfer by paths 248.

20 Server module B 204 illustrates the role of the repeater module 262 in the context
21 of a server module that receives availability information from the initial repeater module
22 A 202. In this role, the message queue 260 of server module B 204 receives the
23 availability information from the message queue 254 of the server module A 202. As
24 described, the server module B 204 can store information in the order it was received, but
25 can override this order in various priority situations. The repeater module 262 of the

1 server module B 204 transfers the availability information received from the message
2 queue 260 into appropriate application caches (230, 232, 234) based on the applications
3 that the availability information pertains to. For example, consider the case where server
4 module A 202 and server module B 204 implement exactly the same applications that
5 access exactly the same resources (in other words, these server modules serve as
6 redundant entities). In this case, the application caches (224, 226, ... 228) of server
7 module A 202 will receive the same availability information as the application caches
8 (230, 232, ... 234) of server module B 204. However, where server module A 202 and
9 server module B 204 do not implement exactly the same applications, the fact that the
10 repeater module 256 of server module A 202 uploads certain availability information to
11 its application caches (224, 226, ... 228) does not necessarily mean that repeater module
12 262 of server module B 204 will do the same with respect to its application caches (230,
13 232, ... 234). The repeater modules implemented on other respective server modules
14 (e.g., 206, ... 208) behave in the manner specified above with respect to repeater module
15 262 of server module B 204.

16 The above description pertained to an implementation in which the initial server
17 module (e.g., server module A 202) broadcasted new availability information to all of the
18 other server modules (e.g., server modules 204, 206, ... 208) in the system 200, as
19 represented by paths 248. However, in another implementation, the system 200 can
20 disseminate new availability information by propagating it from one server module to the
21 next until all of the server modules have received the availability information. That is,
22 this implementation could operate by using the message queue 254 of the initial server
23 module A 202 to send the availability information to the message queue 260 of the next
24 server module B 204, and then the message queue 260 of the server module B 204 could
25

1 transfer the availability information to the message queue (not shown) of server module
2 C 206, and so on. Still other dissemination strategies can be implemented.

3 Finally, the synchronization modules (250, 252, ...) include merge logic (258,
4 264, ...). The function of this logic (258, 264, ...) is explained below in the following
5 section.

6 *Functionality for Synchronizing Availability Information When a Server Module*
7 *Resumes Operation after Having Been Inactive*

8 The above section set forth a procedure whereby new availability information is
9 propagating through the system 200 when it is initially introduced into the system 200
10 from the notification module 242. This serves to synchronize the application caches
11 (224, ... 234) of all of the server modules (202, 204, 206, ... 208) so that they all have
12 the most current availability information. However, for various reasons, one or more of
13 the server modules (202, 204, 206, ... 208) may stop working in a normal manner for a
14 period of time. This event may be attributed to a failure in this server module, or may be
15 attributed to an operator taking this server module offline to perform various
16 maintenance, upgrading, testing, etc. of this server module. Or an operator may add one
17 or more new server modules to the server farm. Whatever the cause, when this server
18 module becomes operational again (or operational for the first time), it may not be in
19 synchronization with the availability information maintained in the other server modules
20 (which have been working continuously). For instance, these other server modules may
21 have been receiving new availability information from the notification module 242. The
22 server module that is being started up (referred to herein as the "starting server module")
23 may not "know" of this information, and is therefore out of synchronization with the
24 other server modules that have been working continuously (referred to herein as the
25 "reference server modules"). The merge logic (258, ...264) addresses this problem.

1 Namely, the merge logic (258, ... 264) can function in two different ways depending on
2 its role in the synchronization operation. The purpose of the merge logic implemented on
3 the starting server module is to discover the availability information stored in the
4 reference server modules and to update its own application caches in accordance
5 therewith. The purpose of the merge logic implemented on the reference server modules
6 is to respond to the inquiries of the starting server module by informing the starting
7 server module of what availability information is new. A server module can function as a
8 starting server module at one instance of time and a reference server module at another
9 instance of time.

10 Fig. 3 illustrates the functions performed by the merge logic (258, ... 264) in
11 greater detail. Fig. 3 again shows the exemplary server modules (202, 204, 206, ... 208)
12 of Fig. 2. Server module A 202 includes synchronization module 250, server module B
13 204 includes synchronization module 252, server module C 206 includes synchronization
14 module 302, and server module n 208 includes synchronization module 304. The
15 contents of the respective synchronization modules (250, 252, 302, ... 304) are simplified
16 to facilitate discussion, showing only the merge logic (258, 264, 306, ... 308) contained
17 therein. In the context of Fig. 3, assume that server module A 202 functions as a so-
18 called starting server module, meaning, as described above, that it has been inactive for
19 some time and that it is now becoming operational. The other server modules (204, 206,
20 ... 208) serve as reference server modules, meaning that these server modules act as the
21 sources from which the starting server module A 202 can discover new availability
22 information that was received by the system 200 during its period of inactivity.

23 As mentioned above, the basic goal of the synchronization procedure is to ensure
24 that the application caches (224, 226, ... 228) of the starting server module A 202 are up
25 to date with respect to the availability information stored in the other reference server

1 modules (204, 206, ... 208). This can be accomplished, broadly speaking, by having the
2 starting server module A 202 make an inquiry to at least one of the reference server
3 modules (204, 206, ... 208). More specifically, assume that the starting server module A
4 202 discovers all of the current availability information maintained by reference server
5 module B 204; then, if server module B 204 has current information, the server module A
6 202 will likewise gain current information upon retrieving this information and loading it
7 into its application caches (224, 226, ... 228). However, depending on the processing
8 environment, there may be a chance that server module B 204 may not have complete up
9 to date availability information. This deficiency may be attributed to the fact that server
10 module B 204 may not have yet received new availability information that is being
11 propagated throughout the system 200 at the time that the starting server A 202 makes its
12 inquiry. In view of this, the starting server module A 202 can also query server module C
13 206 to determine this server module's availability information. Reference server module
14 C 206 can potentially supply availability information that was not conveyed by reference
15 server module B 204. This process can be repeated for additional reference server
16 modules, each time increasing the probability that the starting server module A 202
17 includes all of the new availability information that may exist in the system 200. The
18 number of reference server modules that a starting serving module is configured to query
19 can vary depending on the characteristics of the processing environment. In one case, the
20 starting server module A 202 can only query one reference server module, and in another
21 case, the starting server module A 202 can query, two, three, etc. reference server
22 modules. In one implementation, the starting server module A 202 can be configured to
23 access a predefined list of reference server modules (204, 206, ... 208) when it becomes
24 active. In another implementation, the starting server module A 202 can be configured to
25

1 randomly select its reference server modules (204, 206, ... 208), or use some other
2 technique to select these reference server modules (204, 206, 208).

3 Having set forth the basic philosophy of the synchronization methodology, the
4 exemplary specifics of this operation will be described. First, upon becoming operational
5 again (or starting up for the first time), the starting server module A 202 will restore the
6 availability information contents of its stores that may have existed at the time of shut
7 down. The starting server A 202 then acts on any transactions that may have been
8 pending at the time it became inactive. This can entail transferring pending availability
9 information from its repeater 256 and then to its application caches (224, 226, ... 228).

10 Then assume that the starting server module A 202 first seeks to discover the
11 availability information that reference server module B 204 contains. It performs this
12 task by sending a message to reference server module B 202. This message may include
13 information that reflects the starting server A 202's "understanding" of the current status
14 of the availability information. Upon receiving this message, the merge logic 260 of
15 server module B 204 seeks to determine whether it has any availability information that is
16 not specified in the availability information received from server module A 202. It can
17 perform this function in different ways. One way is to merge the set of availability
18 information received from server module A 202 with the set of availability information
19 currently maintained by server B 204 to form a union set that comprises all the entries in
20 both sets of availability information, but which removes duplicate entries. This union set
21 (i.e., $A \cup B$) can then be transmitted back to the merge logic 258 of server module A 202,
22 where this union set is merged with the availability information currently maintained by
23 server module A 202. The result of this operation therefore yields all of the current
24 availability information according to the universe of server modules defined by at least
25 server module A 202 and server module B 204. The server module A 202 can then act on

1 this updated information by loading it into its respective application caches (224, 226, ...
2 228). At this juncture, the state of the availability information in server module A 202
3 can be represented as $A' = A \cup B$.

4 As described above, gleaning new availability information from reference server
5 module B 204 has a high likelihood of capturing the most current state of the system 200
6 (e.g., assuming that reference server module B 204 was active while starting server
7 module A 202 was inactive). But depending on the type of environment, this likelihood
8 may not be 100 percent. To increase the likelihood of capturing the most current state,
9 the starting server A 202 can repeat the above-described process using server module C
10 206, which prompts server module C 206 to respond with availability information
11 defined by the union of the set of availability information currently maintained by server
12 module A 202 and the set of availability information maintained by server module C 206
13 (i.e., $A' \cup C$). Updating server module A 202 to incorporate this union yields an
14 availability state $A'' = A' \cup C$, which is equivalent to $(A \cup B) \cup C$. If this process is
15 repeated with respect to server module n 208, the server module n 208 will return the
16 response $A'' \cup n$, which is equivalent to $((A \cup B) \cup C) \cup n$, and the likelihood that the
17 starting server module A 202 then contains the most current information maintained in
18 the system 200 will become even greater. Effectively, it can be said that the starting
19 server module A 202 passes all of the availability information that it knows about upon
20 each successive query to a reference server module.

21 In general, the system 204 can be configured to perform the above-described
22 operations in series for reference server module B 204, then reference server module C
23 206, and so on. Alternatively, server module A 202 can broadcast its inquiries to all of
24 the reference server modules (204, 206, ... 208) at the same time, whereupon the merging
25 performed by these reference server modules (204, 206, 208) can occur in parallel.

1 The starting server module A 202 can then also receive responses back from the
2 respective reference server modules (204, 206, ... 208) in parallel. The collected
3 responses can be used to update the starting server module A 202.

4 In addition, the reference server modules (204, 206, ... 208) can also update their
5 application caches (230, 232, ... 234) on the basis of availability information received
6 from the starting server module A 202, which can also incorporate availability
7 information collected from other server modules (e.g., as in the case of availability set A',
8 A'', etc. discussed above). That is, while the main goal of the synchronization procedure
9 is to bring the application caches (224, 226, ... 228) of the starting server module A 202
10 up to date with respect to availability information maintained by reference server
11 modules (204, 206, ... 208), it is also useful to update the reference server modules (204,
12 206, ... 208) based on availability information that these reference server modules (204,
13 206, ... 208) obtain from the starting server module A 202. This is because there is a
14 chance that the starting server module A 202 may include some availability information
15 that is not shared by at least one other reference server module. Thus, when the reference
16 server modules (204, 206, ... 208) perform a merge based on the availability information
17 obtained from the starting server module A 202, they can also use the resultant union set
18 to update their own application caches (e.g., 230, 232, ... 234).

19 By virtue of the above-described synchronization procedure, server module n 208
20 can be assured to receive all of the current availability information offered by server
21 modules B 204 and C 206 (because server module n 208 has received $A'' = A' \cup C$, which
22 is equivalent to $(A \cup B) \cup C$). However, suppose that server module C 206 or server
23 module n 208 offered new availability information when they were queried. In this case,
24 the algorithm described above would not propagate this information to server module B
25 204 (because such information was discovered by server module A 202 after it had

1 finished communicating with server module B 204). To address this situation, server
2 module A 202 can be configured to keep track of those server modules that have not been
3 notified of new availability information. After server module A 202 finishes investigating
4 each of the server modules in sequence, it can then determine whether some of the server
5 modules may have missed out on receiving updated availability information (because of
6 their “position” within the polling sequence). It can then notify each of these server
7 modules of the availability information that they may have missed.

8 The merge logic (258, 264, 306, ... 308) can perform its merging functionality by
9 storing the respective sets of availability information in a store that is idempotent. If an
10 idempotent store receives the same entry twice, it will maintain only one version of this
11 entry. Accordingly, if set (a, b, c, d) is merged with set (c, d, e, f), the result will be (a, b,
12 c, d, e, f), because the idempotent store will keep only one copy of entries c and d.

13 The above-described technique for alerting the starting server module A 202 of the
14 existence of new availability information maintained by other reference server modules
15 (204, 206, ... 208) is only one exemplary technique for performing this operation; other
16 techniques can be used.

17 18 **B. Exemplary Method of Operation**

19 The flow charts in Figs. 4-7 summarize the functionality described in Section A.
20 In general, to facilitate discussion, certain operations are described as constituting distinct
21 steps performed in a certain order. * Such implementations are exemplary and non-
22 limiting. Certain steps described herein can be grouped together and performed in a
23 single operation, and certain steps can be performed in an order that differs from the
24 order employed in the examples set forth in this disclosure.

25 *Determining Application Availability Using Application Caches*

1 To begin with, Fig. 4 describes a procedure 400 for using the system 200 of Fig. 2
2 to respond to the user 218's request for services. In step 402, the user 218 makes an
3 inquiry to the system 200, e.g., using requesting module 220 (which may comprise a
4 personal computer or an application-specific device). In step 408, the request is routed to
5 one of the server modules (202, 204, 206, ... 208) in the system. This routing can be
6 made based on addressing information in the request itself and/or based on load-
7 balancing considerations. In step 406, the allocated server module determines the
8 availability of the application invoked by the request by querying the application cache
9 associated with this application. The information in the application cache defines
10 whether the application is available or unavailable. If it is unavailable, the information in
11 the application cache may also provide reasons why it is unavailable (and optionally can
12 include additional supplemental information regarding the unavailable status). In step
13 408, the appropriate server module performs the application (if it is available) or sends an
14 error message to the user 218 alerting the user 218 of the unavailability of the application
15 (and optionally, the reasons why the application is unavailable, etc.).

16 As mentioned above, the use of the application caches (224, ... 234) to alert the
17 user 218 to the unavailability of applications improves the user 218's experience in
18 interacting with the system 200. For instance, due to the use of local application caches,
19 the user 218 is quickly notified of unavailable applications, thus reducing the chance that
20 the user 218 will waste efforts interacting with the system 200 only to later find out that
21 he or she cannot successfully complete a transaction because some system resource is
22 inactive. The use of availability information provides other benefits, which were set forth
23 in the preceding section.

24 *Disseminating Resource Information to the Distributed Application Caches*

25

1 Fig. 5 describes a procedure 500 for disseminating availability information from a
2 server module that initially receives new availability information from the notification
3 module 242 to the other server modules in the system 200. In the context of Fig. 2, this
4 procedure entails disseminating information first received at server module A 202
5 (referred to as the initial server module) to the other sever modules (204, 206, ... 208).

6 In step 502, the procedure 500 determines whether the status of any application in
7 the system 200 has changed. This can correspond to the case where an operator 240 has
8 entered new availability information into the notification module 242 or the system
9 monitoring functionality 244 has detected a change in the system 200, which, in either
10 case, causes the notification module 242 to transmit a message to the message queue 254
11 of the initial server module A 202. In step 504, the initial server module A 202 loads this
12 new availability information into its own applications caches (224, 226, ... 228) using its
13 repeater module 256. In step 506, the message queue 254 of the initial server module A
14 202 transfers the new availability information to the respective message queues (260, ...)
15 of the other server modules (204, 206, ... 208). This transfer can take place in parallel,
16 where the initial server module A 202 broadcasts the new availability information to all
17 of the other server modules (204 206, ... 208) at the same time, or it can occur in series,
18 where one server module is updated after another in series. In any event, in step 508, the
19 other server modules (204, 206, ... 208) act on the new availability information by
20 loading it from their respective message queues (260, ...) into their respective application
21 caches (230, 232, ...) using their respective repeater modules (262, ...). In general, each
22 repeater is configured to distinguish update messages it receives from "outside" sources
23 (such as the notification module 242) from messages that it receives from other repeaters,
24 and process these messages accordingly.

*Synchronizing Availability Information When a Server Module Resumes
Operation After Having Been Inactive*

Figs. 6 and 7 describe two procedures (600, 700) for synchronizing the system 200 when one of the server modules (202, 204, 206, ... 208) resumes operation after it has been inactive for any reason, or starts up for the first time. For instance, the inactive server module may have been rendered inoperable because of a failure of some sort, or it may have been rendered inoperable because it was deliberately brought offline to perform maintenance, upgrading, testing. More specifically, Fig. 6 addresses the part of the synchronization operation from the vantage point of the server module that is being started up (again referred to as the “starting server module”), while Fig. 7 addresses the part of the synchronization operation from the vantage point of the other server modules which serve as references to the starting server module by supplying current availability information to the starting server module (where these other server modules are again referred to as “reference server modules”). In the context of Fig. 3, the server module A 202 functions as the starting server module, and the other server modules (204, 206, ... 208) function as reference server modules. This arbitrary scenario will serve as the basis of the discussion below.

To begin with, step 602 of Fig. 6 involves determining whether a server module needs updating. This can be satisfied when, as described above, a starting server module has been inactive for any reason, and it is detected that it now wishes to become operational again. The system 200 can detect this event by virtue of the starting server module sending a predetermined startup signal to the other server modules, or by virtue of some other mechanism. In step 604, the starting server module – in this case starting server module A 202 – receives update information from one or more other reference server modules (204, 206, ... 208). As described in connection with Fig. 3 above, this

1 operation can entail transmitting the set of availability information maintained by the
2 starting server module A 202 to a reference server module (e.g., reference server module
3 B 204), merging the set of availability information received from the starting server
4 module A 202 with the set of availability information maintained by the reference server
5 module B 204 to provide a union set, and then transmitting this union set back to the
6 starting server module A 202. In step 606, the starting server module A 202 receives this
7 union set and merges it with its current set of availability information. This procedure
8 can be performed once with respect to one reference server module, or can be performed
9 with respect to multiple reference server modules. In any case, the merging can be
10 implemented by feeding different sets of availability information maintained by different
11 server modules into an idempotent store. If the idempotent store receives a copy of an
12 entry that it already contains, it will store only one copy of this entry. That is, it will not
13 store duplicate entries.

14 Procedure 700 of Fig. 7 describes the synchronization procedure from the vantage
15 point of a reference server module, e.g., reference server module B 204. In step 702, the
16 reference server module B 204 receives availability information from the starting server
17 module A 202. In step 704, the reference server module B 204 merges the set of
18 availability information obtained from the starting server module A 202 with the set of
19 availability information obtained by the reference server module B 204 to provide a
20 union set. In step 706, the reference server module B 204 sends the union set to the
21 starting server module A 202, whereupon the starting server module A 202 loads any new
22 information contained therein in its application caches (224, 226, ... 228). (Alternatively,
23 if the reference server module B 204 determines that the union set $A \cup B$ does not contain
24 any information beyond that specified in set A, it need not send a message back to the
25 starting server module A 202.) In step 708, the reference server module B 204 can

1 update its own application caches (230, 232, ... 234) based on any new information that
2 it may have gleaned from the starting server module A 202. That is, while the main
3 purpose the synchronization procedure is to bring the starting server module A 202 up to
4 date with respect to current availability information in the system 200, there is a chance
5 that the starting server module A 202 may having availability information that is new to
6 the reference server module B 204. Step 708 accounts for this possibility by providing a
7 mechanism for updating the application caches (230, 232, ... 234) of the reference server
8 module B 204. Although not shown in Fig. 7, the reference server module B 204 may
9 also later receive a follow up update from the starting server module A 202. This follow
10 up update may include any new availability information that the starting server module A
11 202 has discovered in the course of querying reference server module C 206, reference
12 server module n 208, and so on.

13
14 In closing, a number of examples were presented in this disclosure in the
15 alternative (e.g., case X or case Y). In addition, this disclosure encompasses those cases
16 which combine alternatives in a single implementation (e.g., case X and case Y), even
17 though this disclosure may have not expressly mentioned these conjunctive cases in every
18 instance.

19 More generally, although the invention has been described in language specific to
20 structural features and/or methodological acts, it is to be understood that the invention
21 defined in the appended claims is not necessarily limited to the specific features or acts
22 described. Rather, the specific features and acts are disclosed as exemplary forms of
23 implementing the claimed invention.